ArduinoISP

Troubleshooting of "programmer is not in sync"

- Try to use different version of Arduino IDE, Arduino IDE 1.0 with UNO will cause this problem.
- Double check all the connections
- The receive side of Arduino is out of buffer, if you are using the official "ArduinoISP" in the Arduino IDE examples folder, please try to use the following code. This will solve the problem too:

```
avrdude.exe: stk500_paged_write(): (a) protocol error, expect=0x14, resp=0x64
avrdude.exe: stk500_cmd(): programmer is out of sync
```

Main Sketch (Retired Code)

- Replace the A0 pin with pin 7, which is programming pin.

```
// this sketch turns the Arduino into a AVRISP
// using the following pins:
// 10: slave reset
// 11: MOSI
// 12: MISO
// 13: SCK

// Put an LED (with resistor) on the following pins:
//  8: Error - Lights up if something goes wrong (use red if that makes sense)
// A0: Programming - In communication with the slave
//  6: Heartbeat - shows the programmer is running (removed, see notes below)
// Optional - Piezo speaker on pin A3
//
// October 2009 by David A. Mellis
// - Added support for the read signature command
//
// February 2009 by Randall Bohn
// - Added support for writing to EEPROM (what took so long?)
// Windows users should consider WinAVR's avrdude instead of the
// avrdude included with Arduino software.
//
// January 2008 by Randall Bohn
// - Thanks to Amplificar for helping me with the STK500 protocol
// - The AVRISP/STK500 (mk I) protocol is used in the arduino bootloader
// - The SPI functions herein were developed for the AVR910_ARD programmer
// - More information at http://code.google.com/p/mega-isp
//
// March 2012 - William Phelps
// modify to work with Arduino IDE 1.0 which has a shorter serial port receive buffer
// getEOP() now gets entire request before avrisp() is called to process it
// Serial.print((char) xxx) changed to Serial.write(xxx)
// uint8_t changed to byte
// added support for Piezo speaker
// moved Pmode LED to A0
```

```
// removed "heartbeat" on pin 6, added short blip of ERROR LED instead
// Why is it that PROG_FLASH and PROG_DATA don't actually do anything???
// Tested with Arduino IDE 22 and 1.0
// IDE 22 - 5148 bytes
// IDE 1.0 - 5524 bytes!


// SLOW SPEED CHIP ERASE AND FUSE BURNING
//
// Enable LOW_SPEED to allow you to erase chips that would fail otherwise,
// for being running with a clock too slow for the programmer.
//
// This allowed me to recover several ATMega328 that had no boot loader and the
// first instruction was to set the clock to the slowest speed. Usually this
// kind of recovery requires high voltage programming, but this trick will do
// just fine.
//
// How to proceed:
// 1. Enable LOW_SPEED, and load it to the programmer.
// 2. Erase and burn the fuses on the target uC. Example for ATMega328:
//   arduino-1.0.1/hardware/tools/avrdude -Carduino-1.0.1/hardware/tools/avrdude.conf -
patmega328p -cstk500v1 -P /dev/serial/by-id/usb-FTDI_FT232R_USB_UART_A900cf1Q-if00-
port0 -b19200 -e -Ulock:w:0x3F:m -Uefuse:w:0x05:m -Uhfuse:w:0xDA:m -Ulfuse:w:0xF7:m
// 3. Comment LOW_SPEED and load it back to the programmer.
// 4. Program the target uC as usual. Example:
//  arduino-1.0.1/hardware/tools/avrdude -Carduino-1.0.1/hardware/tools/avrdude.conf -
patmega328p -cstk500v1 -P /dev/serial/by-id/usb-FTDI_FT232R_USB_UART_A900cf1Q-if00-
port0 -b19200 -Uflash:w:firmware.hex:i
//
// Note 1: EXTRA_SPI_DELAY was added to let you slow down SPI even more. You can
// play with the value if it does not work with the default.
// Note 2: LOW_SPEED will alow you only to erase the chip and burn the fuses! It
// will fail if you try to program the target uC this way!

//#define LOW_SPEED
#ifdef LOW_SPEED
#define EXTRA_SPI_DELAY 125
#else
#define EXTRA_SPI_DELAY 0
#endif

#include "pins_arduino.h"  // defines SS,MOSI,MISO,SCK
#define RESET SS

#define LED_ERR 8
#define LED_PMODE A0
//#define LED_HB 6
#define PIEZO A3
```

```
#define HWVER 2
#define SWMAJ 1
#define SWMIN 18

// STK Definitions
const byte STK_OK = 0x10;
const byte STK_FAILED = 0x11;
const byte STK_UNKNOWN = 0x12;
const byte STK_INSYNC = 0x14;
const byte STK_NOSYNC = 0x15;
const byte CRC_EOP = 0x20; //ok it is a space...

const byte STK_GET_SYNC     = 0x30;
const byte STK_GET_SIGNON   = 0x31;
const byte STK_GET_PARM     = 0x41;
const byte STK_SET_PARM     = 0x42;
const byte STK_SET_PARM_EXT = 0x45;
const byte STK_PMODE_START  = 0x50;
const byte STK_PMODE_END    = 0x51;
const byte STK_SET_ADDR     = 0x55;
const byte STK_UNIVERSAL    = 0x56;
const byte STK_PROG_FLASH   = 0x60;
const byte STK_PROG_DATA    = 0x61;
const byte STK_PROG_PAGE    = 0x64;
const byte STK_READ_PAGE    = 0x74;
const byte STK_READ_SIGN    = 0x75;

//// TONES  ======================================
//// Start by defining the relationship between
////       note, period, &  frequency.
//#define  c     3830    // 261 Hz
//#define  d     3400    // 294 Hz
//#define  e     3038    // 329 Hz
//#define  f     2864    // 349 Hz
//#define  g     2550    // 392 Hz
//#define  a     2272    // 440 Hz
//#define  b     2028    // 493 Hz
//#define  C     1912    // 523 Hz

//void pulse(int pin, int times);

int error=0;
int pmode=0;
// address for reading and writing, set by STK_SET_ADDR command
int _addr;
byte _buffer[256]; // serial port buffer
int pBuffer = 0;  // buffer pointer
int iBuffer = 0;  // buffer index
byte buff[256];  // temporary buffer
```

```
boolean EOP_SEEN = false;

void setup() {

  Serial.begin(19200);
  pinMode(PIEZO, OUTPUT);
  beep(1700, 40);
  EOP_SEEN = false;
  iBuffer = pBuffer = 0;

  pinMode(LED_PMODE, OUTPUT);
  pulse(LED_PMODE, 2);
  pinMode(LED_ERR, OUTPUT);
  pulse(LED_ERR, 2);
//  pinMode(LED_HB, OUTPUT);
//  pulse(LED_HB, 2);

  pinMode(9, OUTPUT);
  // setup high freq PWM on pin 9 (timer 1)
  // 50% duty cycle -> 8 MHz
  OCR1A = 0;
  ICR1 = 1;
  // OC1A output, fast PWM
  TCCR1A = _BV(WGM11) | _BV(COM1A1);
  TCCR1B = _BV(WGM13) | _BV(WGM12) | _BV(CS10); // no clock prescale


}

#define beget16(addr) (*addr * 256 + *(addr+1) )
typedef struct param {
  byte devicecode;
  byte revision;
  byte progtype;
  byte parmode;
  byte polling;
  byte selftimed;
  byte lockbytes;
  byte fusebytes;
  int flashpoll;
  int eeprompoll;
  int pagesize;
  int eepromsize;
  int flashsize;
}
parameter;

parameter param;
```

```
// this provides a heartbeat on pin 6, so you can tell the software is running.
//byte hbval=128;
//int8_t hbdelta=4;
//void heartbeat() {
////  if (hbval > 192) hbdelta = -hbdelta;
////  if (hbval < 32) hbdelta = -hbdelta;
//  if (hbval > 250) hbdelta = -hbdelta;
//  if (hbval < 10) hbdelta = -hbdelta;
//  hbval += hbdelta;
//  analogWrite(LED_HB, hbval);
//  delay(20);
//}

void getEOP() {
  int minL = 0;
  byte avrch = 0;
  byte bl = 0;
  while (!EOP_SEEN) {
    while (Serial.available()>0) {
      byte ch = Serial.read();
      _buffer[iBuffer] = ch;
      iBuffer = (++iBuffer)%256;  // increment and wrap
      if (iBuffer == 1)  avrch = ch;  // save command
      if ((avrch == STK_PROG_PAGE) && (iBuffer==3)) {
        minL = 256*_buffer[1] + _buffer[2] + 4;
      }
      if ((iBuffer>minL) && (ch == CRC_EOP)) {
        EOP_SEEN = true;
      }
    }
    if (!EOP_SEEN) {
//      heartbeat(); // light the heartbeat LED
      if (bl == 100) {
        pulse(LED_ERR,1,10);  // blink the red LED
        bl = 0;
      }
      bl++;
      delay(10);
    }
  }
}

// serialEvent not used so sketch would be compatible with older IDE versions
//void serialEvent() {
//  int minL = 0;
//  byte avrch = 0;
//  while (Serial.available()>0)
//  {
//    byte ch = Serial.read();
```

```
//    _buffer[iBuffer] = ch;
//    iBuffer = (++iBuffer)%256;  // increment and wrap
//    if (iBuffer == 1)  avrch = ch;  // save command
//    if ((avrch == STK_PROG_PAGE) && (iBuffer==3)) {
//      minL = 256*_buffer[1] + _buffer[2] + 4;
//    }
//    if ((iBuffer>minL) && (ch == CRC_EOP)) {
//      EOP_SEEN = true;
//    }
//  }
//}

void loop(void) {
  // is pmode active?
//  if (pmode) digitalWrite(LED_PMODE, HIGH);
//  else digitalWrite(LED_PMODE, LOW);
  digitalWrite(LED_PMODE, LOW);
  // is there an error?
  if (error) digitalWrite(LED_ERR, HIGH);
  else digitalWrite(LED_ERR, LOW);

  getEOP();

  // have we received a complete request?  (ends with CRC_EOP)
  if (EOP_SEEN) {
    digitalWrite(LED_PMODE, HIGH);
    EOP_SEEN = false;
    avrisp();
    iBuffer = pBuffer = 0;  // restart buffer
  }

}

byte getch() {
  if (pBuffer == iBuffer) {  // spin until data available ???
    pulse(LED_ERR, 1);
    beep(1700, 20);
    error++;
    return -1;
  }
  byte ch = _buffer[pBuffer];  // get next char
  pBuffer = (++pBuffer)%256;  // increment and wrap
  return ch;
}

void readbytes(int n) {
  for (int x = 0; x < n; x++) {
    buff[x] = getch();
  }
```

```
}

//#define PTIME 20
void pulse(int pin, int times, int ptime) {
  do {
    digitalWrite(pin, HIGH);
    delay(ptime);
    digitalWrite(pin, LOW);
    delay(ptime);
    times--;
  }
  while (times > 0);
}
void pulse(int pin, int times) {
  pulse(pin, times, 50);
}

void spi_init() {
  byte x;
  SPCR = 0x53;
#ifdef LOW_SPEED
SPCR=SPCR|B00000011;
#endif
  x=SPSR;
  x=SPDR;
}

void spi_wait() {
  do {
  }
  while (!(SPSR & (1 << SPIF)));
}

byte spi_send(byte b) {
  byte reply;
#ifdef LOW_SPEED
    cli();
    CLKPR=B10000000;
    CLKPR=B00000011;
    sei();
#endif
  SPDR=b;
  spi_wait();
  reply = SPDR;
#ifdef LOW_SPEED
    cli();
    CLKPR=B10000000;
    CLKPR=B00000000;
   sei();
```

```
#endif
    return reply;
}

byte spi_transaction(byte a, byte b, byte c, byte d) {
  byte n;
  spi_send(a);
  n=spi_send(b);
  //if (n != a) error = -1;
  n=spi_send(c);
  return spi_send(d);
}

void replyOK() {
//  if (EOP_SEEN == true) {
  if (CRC_EOP == getch()) {  // EOP should be next char
    Serial.write(STK_INSYNC);
    Serial.write(STK_OK);
  }
  else {
    pulse(LED_ERR, 2);
    Serial.write(STK_NOSYNC);
    error++;
  }
}

void breply(byte b) {
  if (CRC_EOP == getch()) {  // EOP should be next char
    Serial.write(STK_INSYNC);
    Serial.write(b);
    Serial.write(STK_OK);
  }
  else {
    Serial.write(STK_NOSYNC);
    error++;
  }
}

void get_parameter(byte c) {
  switch(c) {
  case 0x80:
    breply(HWVER);
    break;
  case 0x81:
    breply(SWMAJ);
    break;
  case 0x82:
    breply(SWMIN);
    break;
```

```
  case 0x93:
    breply('S'); // serial programmer
    break;
  default:
    breply(0);
  }
}

void set_parameters() {
  // call this after reading paramter packet into buff[]
  param.devicecode = buff[0];
  param.revision = buff[1];
  param.progtype = buff[2];
  param.parmode = buff[3];
  param.polling = buff[4];
  param.selftimed = buff[5];
  param.lockbytes = buff[6];
  param.fusebytes = buff[7];
  param.flashpoll = buff[8];
  // ignore buff[9] (= buff[8])
  //getch(); // discard second value

  // WARNING: not sure about the byte order of the following
  // following are 16 bits (big endian)
  param.eeprompoll = beget16(&buff[10]);
  param.pagesize = beget16(&buff[12]);
  param.eepromsize = beget16(&buff[14]);

  // 32 bits flashsize (big endian)
  param.flashsize = buff[16] * 0x01000000
    + buff[17] * 0x00010000
    + buff[18] * 0x00000100
    + buff[19];

}

void start_pmode() {
  spi_init();
  // following delays may not work on all targets...
  pinMode(RESET, OUTPUT);
  digitalWrite(RESET, HIGH);
  pinMode(SCK, OUTPUT);
  digitalWrite(SCK, LOW);
  delay(50+EXTRA_SPI_DELAY);
  digitalWrite(RESET, LOW);
  delay(50+EXTRA_SPI_DELAY);
  pinMode(MISO, INPUT);
  pinMode(MOSI, OUTPUT);
  spi_transaction(0xAC, 0x53, 0x00, 0x00);
```

```
  pmode = 1;
}

void end_pmode() {
  pinMode(MISO, INPUT);
  pinMode(MOSI, INPUT);
  pinMode(SCK, INPUT);
  pinMode(RESET, INPUT);
  pmode = 0;
}

void universal() {
//  int w;
  byte ch;
//  for (w = 0; w < 4; w++) {
//    buff[w] = getch();
//  }
  readbytes(4);
  ch = spi_transaction(buff[0], buff[1], buff[2], buff[3]);
  breply(ch);
}

void flash(byte hilo, int addr, byte data) {
  spi_transaction(0x40+8*hilo, addr>>8 & 0xFF, addr & 0xFF, data);
}
void commit(int addr) {
  spi_transaction(0x4C, (addr >> 8) & 0xFF, addr & 0xFF, 0);
}

//#define _current_page(x) (here & 0xFFFFE0)
int current_page(int addr) {
  if (param.pagesize == 32) return addr & 0xFFFFFFF0;
  if (param.pagesize == 64) return addr & 0xFFFFFFE0;
  if (param.pagesize == 128) return addr & 0xFFFFFFC0;
  if (param.pagesize == 256) return addr & 0xFFFFFF80;
  return addr;
}
byte write_flash(int length) {
  if (param.pagesize < 1) {
    return STK_FAILED;
  }
  //if (param.pagesize != 64) return STK_FAILED;
  int page = current_page(_addr);
  int x = 0;
  while (x < length) {
    if (page != current_page(_addr)) {
      commit(page);
      page = current_page(_addr);
    }
```

```
    flash(LOW, _addr, buff[x++]);
    flash(HIGH, _addr, buff[x++]);
    _addr++;
  }
  commit(page);
  return STK_OK;
}

byte write_eeprom(int length) {
  // here is a word address, so we use here*2
  // this writes byte-by-byte,
  // page writing may be faster (4 bytes at a time)
  for (int x = 0; x < length; x++) {
    spi_transaction(0xC0, 0x00, _addr*2+x, buff[x]);
    delay(45);
  }
  return STK_OK;
}

void program_page() {
  byte result = STK_FAILED;
  int length = 256 * getch() + getch();
  if (length > 256) {
      Serial.write(STK_FAILED);
      error++;
      return;
  }
  char memtype = (char)getch();
//  for (int x = 0; x < length; x++) {
//    buff[x] = getch();
//  }
  readbytes(length);
  if (CRC_EOP == getch()) {
    Serial.write(STK_INSYNC);
    switch (memtype) {
      case 'E':
        result = (byte)write_eeprom(length);
        break;
      case 'F':
        result = (byte)write_flash(length);
        break;
    }
    Serial.write(result);
    if (result != STK_OK) {
      error++;
    }
  }
  else {
    Serial.write(STK_NOSYNC);
```

```
    error++;
  }
}

byte flash_read(byte hilo, int addr) {
  return spi_transaction(0x20 + hilo * 8,
    (addr >> 8) & 0xFF,
    addr & 0xFF,
    0);
}

char flash_read_page(int length) {
  for (int x = 0; x < length; x+=2) {
    byte low = flash_read(LOW, _addr);
    Serial.write( low);
    byte high = flash_read(HIGH, _addr);
    Serial.write( high);
    _addr++;
  }
  return STK_OK;
}

char eeprom_read_page(int length) {
  // here again we have a word address
  for (int x = 0; x < length; x++) {
    byte ee = spi_transaction(0xA0, 0x00, _addr*2+x, 0xFF);
    Serial.write( ee);
  }
  return STK_OK;
}

void read_page() {
  byte result = (byte)STK_FAILED;
  int length = 256 * getch() + getch();
  char memtype = getch();
  if (CRC_EOP != getch()) {
    Serial.write(STK_NOSYNC);
    return;
  }
  Serial.write(STK_INSYNC);
  if (memtype == 'F') result = flash_read_page(length);
  if (memtype == 'E') result = eeprom_read_page(length);
  Serial.write(result);
  return;
}

void read_signature() {
  if (CRC_EOP != getch()) {
    Serial.write(STK_NOSYNC);
```

```
    error++;
    return;
  }
  Serial.write(STK_INSYNC);
  byte high = spi_transaction(0x30, 0x00, 0x00, 0x00);
  Serial.write(high);
  byte middle = spi_transaction(0x30, 0x00, 0x01, 0x00);
  Serial.write(middle);
  byte low = spi_transaction(0x30, 0x00, 0x02, 0x00);
  Serial.write(low);
  Serial.write(STK_OK);
}
////////////////////////////////////////
////////////////////////////////////////


////////////////////////////////////
////////////////////////////////////

int avrisp() {
  byte data, low, high;
  byte avrch = getch();
  switch (avrch) {
    case STK_GET_SYNC: // get in sync
      replyOK();
      break;
    case STK_GET_SIGNON:  // get sign on
      if (getch() == CRC_EOP) {
        Serial.write(STK_INSYNC);
        Serial.write("AVR ISP");
        Serial.write(STK_OK);
      }
      break;
    case STK_GET_PARM:  // 0x41
      get_parameter(getch());
      break;
    case STK_SET_PARM:  // 0x42
      readbytes(20);
      set_parameters();
      replyOK();
      break;
    case STK_SET_PARM_EXT: // extended parameters - ignore for now
      readbytes(5);
      replyOK();
      break;
    case STK_PMODE_START:  // 0x50
      beep(2272, 20);
      start_pmode();
      replyOK();
```

```
      break;
    case STK_PMODE_END:  //0x51
      beep(1912, 50);
      error=0;
      end_pmode();
      replyOK();
      break;
    case STK_SET_ADDR:  // 0x55
      _addr = getch() + 256 * getch();
      replyOK();
      break;
    case STK_UNIVERSAL:  //UNIVERSAL 0x56
      universal();
      break;
    case STK_PROG_FLASH: //STK_PROG_FLASH  ???
      low = getch();
      high = getch();
      replyOK();
      break;
    case STK_PROG_DATA: //STK_PROG_DATA  ???
      data = getch();
      replyOK();
      break;
    case STK_PROG_PAGE:  //STK_PROG_PAGE
//      beep(1912, 20);
      program_page();
      break;
    case STK_READ_PAGE: //STK_READ_PAGE
      read_page();
      break;
    case STK_READ_SIGN: //STK_READ_SIGN
      read_signature();
      break;
    // expecting a command, not CRC_EOP
    // this is how we can get back in sync
    case CRC_EOP:
      Serial.write(STK_NOSYNC);
      break;
    // anything else we will return STK_UNKNOWN
    default:
      if (CRC_EOP == getch())
        Serial.write(STK_UNKNOWN);
      else
        Serial.write(STK_NOSYNC);
  }
}

// beep without using PWM
void beep(int tone, long duration){
```

```
  long elapsed = 0;
  while (elapsed < (duration * 10000)) {
    digitalWrite(PIEZO, HIGH);
    delayMicroseconds(tone / 2);
    digitalWrite(PIEZO, LOW);
    delayMicroseconds(tone / 2);
    // Keep track of how long we pulsed
    elapsed += tone;
  }
}
```